目录

[1.需求分析 1](#_Toc39856681)

[2 软件框架 2](#_Toc39856682)

[3 模块详解 4](#_Toc39856683)

[3.1 数据类型介绍 4](#_Toc39856684)

[3.2初始化模块 5](#_Toc39856685)

[3.3联结功能模块 5](#_Toc39856686)

[3.4分组功能模块 6](#_Toc39856687)

[3.5排序功能模块 6](#_Toc39856688)

[4 结果分析 6](#_Toc39856689)

[4.1 算法复杂度分析 6](#_Toc39856690)

[5 经验总结 6](#_Toc39856691)

## 1.需求分析

#### 1.1题目要求

现有两张数据表t1(id1 int primary key,id2 int,id3 int)、t2(id1 int primary key,id2 int,id3 int)，其中表t1的数据存储在文件input1.csv中，表t2的数据存储在文件input2.csv中。每个文件中有三个整形字段，字段之间用逗号分开，记录条目之间换行。要求针对以上给出的t1、t2两个表数据，采用C/C++开发能模拟数据库连接(JOIN)、分组(GROUP BY)、排序(ORDER BY)功能的算法程序，正确计算出下面语句的结果：

select max(t1.id1),min(t2.id1) from t1 join t2 on t1.id3 = t2.id3 group by t1.id2,t2.id2 order by max(t1.id1),t2.id2,t1.id2;

#### 1.2额外信息

a. 算法所使用的内存不超过20M

b. 输入文件路径为：/home/web/ztedatabase/input1.csv和/home/web/ztedatabase/input2.csv

c. 输出请输出到"标准系统输出"中

#### 1.3验收标准

a. 输出结果正确

b. 时间越快越优

c. 内存控制合理

#### 1.4举例说明

##### 1.4.1表信息

Tbale1:

|  |  |  |  |
| --- | --- | --- | --- |
| index | Id1 | Id2 | Id3 |
| a1 | 1 | 1 | 1 |
| a2 | 2 | 0 | 1 |
| a3 | 3 | 4 | 2 |
| a4 | 4 | 2 | 1 |
| a5 | 5 | 2 | 3 |

Tbale2:

|  |  |  |  |
| --- | --- | --- | --- |
| index | Id1 | Id2 | Id3 |
| b1 | 1 | 3 | 3 |
| b2 | 2 | 0 | 0 |
| b3 | 3 | 0 | 1 |
| b4 | 4 | 0 | 1 |
| b5 | 5 | 4 | 2 |

##### 1.4.2 Join结果

Select \* from t1 join t2 on t1.id3=t2.id3;（根据id3字段将t1表join至t2表合成一个新表）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| index | t1.id1 | t1.id2 | t1.id3 | t1.id1 | t1.id2 | t1.id3 |
| a1-b3 | 1 | 1 | 1 | 3 | 0 | 1 |
| a2-b3 | 2 | 0 | 1 | 3 | 0 | 1 |
| a4-b3 | 4 | 2 | 1 | 3 | 0 | 1 |
| a1-b4 | 1 | 1 | 1 | 4 | 0 | 1 |
| a2-b4 | 2 | 0 | 1 | 4 | 0 | 1 |
| a4-b4 | 4 | 2 | 1 | 4 | 0 | 1 |
| a3-b5 | 3 | 4 | 2 | 5 | 4 | 2 |
| a5-b1 | 5 | 2 | 3 | 1 | 3 | 3 |

##### 1.4.3 group结果

select t1.id1, t1.id2, t2.id1, t2.id2 from t1 join t2 on t1.id3=t2.id3 group by t1.id2, t2.id2;

(在上述结果的基础上进行分组，其中t1.id2以及t2.id2相同则分为一组，此时我们就不在关注id3这个字段，所以显示的结果中屏蔽了id3字段，该图只是为了说明运算过程，而非实际数据库语句结果图).



##### 1.4.4.order结果

select t1.id1, t1.id2, t2.id1, t2.id2 from t1 join t2 on t1.id3=t2.id3 **group by** t1.id2, t2.id2 **order by** max(t1.id1), t2.id2, t1.id2;

最后一个阶段就是排序了，排序的字段按照**每一组的t1.id1的最大值，t2.id2, t1.id2进行。**得到的结果如图所示（该图只是为了说明运算过程，而非实际数据库语句结果图）：



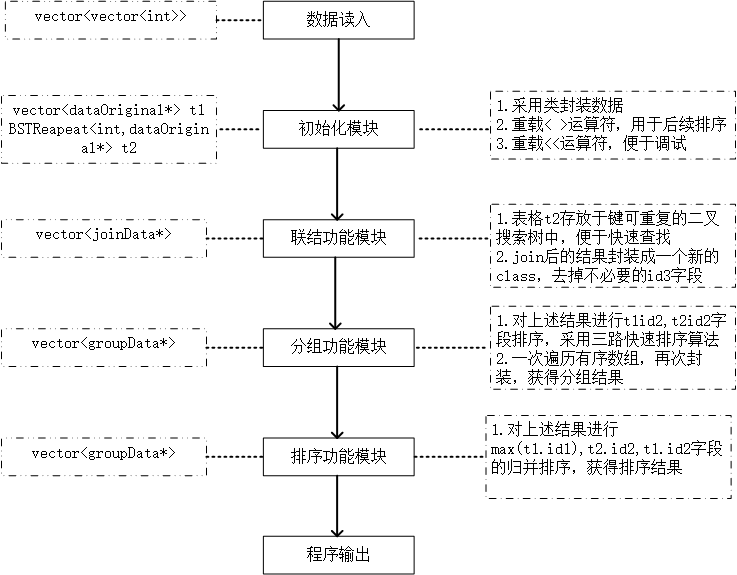
##### 1.4.5程序输出结果

在上述结果基础上，程序需要输出每一组max(t1.id1)和min(t2.id1)就可。



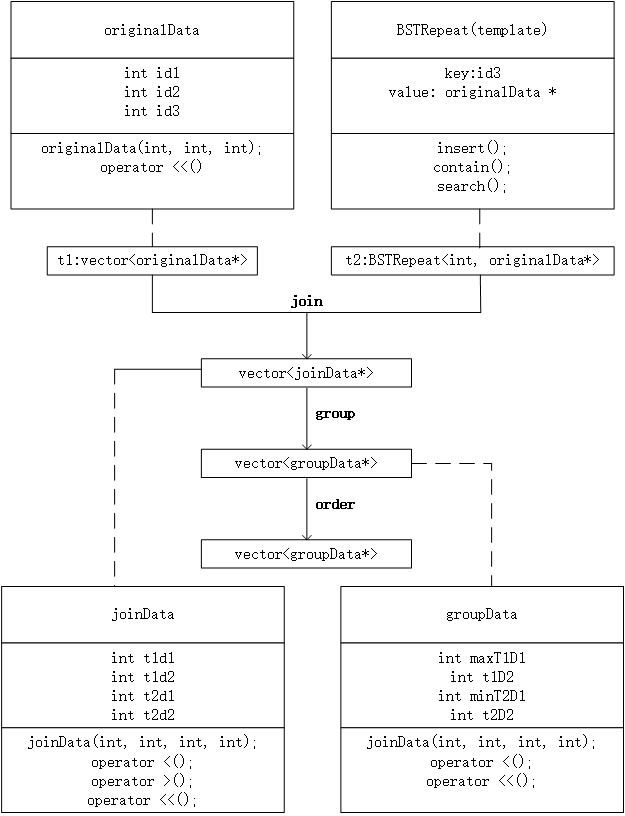
## 2 软件框架

整体算法框架见图2-1



## 3 模块详解

### 3.1 数据类型介绍



a. 数据读入阶段：采用两个vector<vector<int>>保存读入的数据

b. 数据初始化阶段：t1表格采用vector<originalData\*>保存，t2表格采用二叉搜索树保存，每一个节点由pair组成，key是t2.id3字段，value是对应的originalData\*型数据。

c. 联结阶段：根据t1.id3和t2.id3相匹配，获得join后的结果，保存至vector<joinData\*>中。其中，joinData去掉了t1,t2中的id3字段信息。

d. 分组阶段：分完组，保存每一个组的max(t1.id1)，t1.id2, min(t2.id1), t2.id2信息。保存至vecotr<groupData\*>中。

e. 排序阶段：直接对上述结果进行组内排序，类型依然是vector<groupData\*>。

### 3.2初始化模块

#### 3.2.1 t1表格初始化

数据读入得到的是vector<vector<int>>类型数据，对每条数据进行实例化（originalData）。最终保存至vector<originalData\*>中。

#### 3.2.2 t2表格初始化

由于需要在表格2中查找id3字段与t1表匹配，将t2中的数据存放于二叉搜索树中，这样可以更加快速的查找数据。t2表格id3字段可能存在重复，因此要求二叉搜索树可以存储重复key节点，对每一个树节点添加一个next指针，如果对应的key重复，添加至对应的链表中。其示意图如图3-2所示。

由于二叉搜索树在不合理的节点分布情况下，可能退化成链表，对应各操作的时间复杂度由原来的O(logn)，增加至O(n)，因此，在初始化t2表格时，对要插入的数据进行一次random\_shuffle，减少退化成链表的可能。但是无法避免左右子树的不平衡。

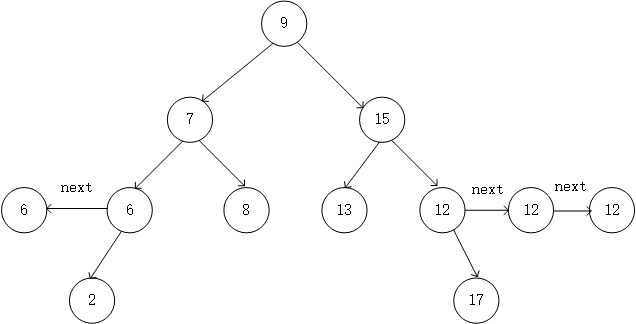


图3-2. 二叉搜索树示意图

### 3.3联结功能模块

遍历一遍t1表格，取当前时刻t1.id3的值，并在二叉搜索树中寻找对应key的节点，如果找到对应的节点，则将该条信息与t1对应的信息融合实例化一个新的类对象(joinData)，加入到一个vector中。算法流程图如图3-3所示

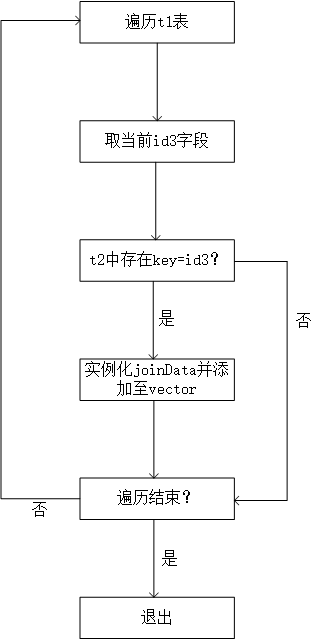


图3-3 join算法流程图

### 3.4分组功能模块

分组是依据t1.id2,t2.id2进行，先对join的结果按照t1.id2,t2.id2字段进行一次排序， 获得按t1.id2, t2.id2有序的结果，然后一次遍历有序数组就可以得到分组的情况。此时，只需要保存每一组的max(t1.id1)，t1.id2, min(t2.id1), t2.id2即可。因为每一组的t1.id2, t2.id2相同。额外加上t1.id1的最大值，t2.id1的最小值。时间复杂度是O(n)。

排序算法采用快速排序，时间复杂度是O(nlogn)。由于是对对象进行排序，对应的类重载了对应的运算符，按照常规比较即可。

### 3.5排序功能模块

最后一部分是对分组后的结果进行一次排序，此时数据类型是groupData，同时重载了对应的运算符（大于号和小于号）记录了每一次的max(t1.id1), t1.id2, min(t2.id1), t2.id2。按照max(t1.id1)和t2.id2以及t1.id2字段排序即可，采用归并排序。时间复杂度是O(nlogn)。

## 4 结果分析

### 4.1 算法复杂度分析

#### 4.1.1时间复杂度分析

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 阶段 | 初始化 | 联结 | 分组 | 排序 | 总计 |
| 理想情况 | O(n) | O(nlogn) | O(nlogn) | O(nlogn) | O(nlogn) |
| 最坏情况 | O(n) | O(n^2) | O(nlogn) | O(nlogn) | O(n^2) |
| 本地测试结果(ms) | 6.46 | 203 | 865(其中排序742) | 14 | 1088 |

从表中数据可看出，分组占用了较长时间，后续优化可从此方面入手。

#### 4.1.2 空间复杂度分析

前期采用stl库中的map和set，超出内存太多，后改用自定义二叉搜索树完成，内存控制在合理范围内。

## 5 经验总结

通过此次比赛，接触了很多东西，总结如下

（1）入门SQL。从开始完全没有接触过SQL，题目都看不懂，到后续自己摸索，安装MySQL和SQLyog学习数据库管理，自己学到了很多，但是同时也明白还有很多东西要学习。

（2）学习在linux环境下测试程序的内存占用情况。程序=数据结构+算法，一个良好的程序不仅在时间上表现优秀，空间上能控制在合理范围也很重要。通过linux环境下的Valgrind工具可详细得到程序运行时程序占用情况，非常的方便和有用。参考链接：https://blog.csdn.net/fengbingchun/article/details/83279163

（3）数据结构真的很强大。实际应用中C++标准模板库已经封装好，直接使用即可。但是这对一个优秀的开发人员远远不够，最开始采用multiset和multimap完成，内存分析后，程序占用几百兆的内存，大部分都是红黑树，令人大吃一惊。后来逐渐了解到红黑树相比较平衡二叉树，二叉搜索树的特性。才知道stl为什么用红黑树实现set和map，至于后续的unorder\_map和unorder\_set是基于哈希表，更需要好好学习了。这次比赛自己实现的带重复键的二叉搜索树还有问题，例如退化成链表。有待研究。

（4）算法也很重要。常见的排序算法，特性，优缺点等等。这次比赛在自己的数据上，由于有很多重复元素，所以采用的三路快速排序进行分组前的排序工作，表现很好，官方的数据集上没有很大提升，需要想想更好的算法解决这个问题。